第2章 ハードウェア補完と不具合予防のためのノウハウを知る

組み込みソフトウェアの定石テクニック集

舘 伸幸，名野 響，木下秀昭，森 孝夫

ここでは，組み込みシステム開発において，機能や性能が足りないがハードウェアによる対策が取れない場合などのソフトウェアによる解決策を提示する．また，ハードウェアの振る舞いを理解した上でソフトウェアがとるべき対応についても述べる．　（編集部）

組み込み制御で用いられるマイコンは，製造コストを考慮すると余分な機能が入っていないものが最適です．通常は，要求仕様を満たしていて，かつ最も規模の小さいものを選択するのが一般的です．しかし時には，敢えて要求仕様を満たしていないマイコンを選定し，足りない機能を設計時の工夫で補うという手法が採られることも少なくありません．あるいは，マイコン選定時には不要と思っていた機能が開発途中に必要になってしまうこともあります．

ここでは，ハードウェア的に足りない機能や性能をソフトウェアで補う方法や，ハードウェアの振る舞いを理解した上でソフトウェアがとるべき対応について，各項目ごとに解説します．

本稿で解説する項目一覧

1. A-D変換速度を上げたい
2. D-Aコンバータが足りない
3. UARTがない
4. UARTの制御信号がない
5. ポートが足りない
6. タイマが足りない
7. チャタリングを回避する
8. スイッチによるカウントアップをいかに実装するか
9. ハードウェアの動作完了を待つ方法
10. 誤動作時にリセットがかかるようにするには
11. 出力ポートの設定で気を付けたいこと

2

KeyWord　A-Dコンバータ，D-Aコンバータ，ポート，ラダー抵抗，ローパス・フィルタ，UART，制御信号，スタート・ビット，ストップ・ビット，マトリクス回路，ダイナミック表示，チャタリング，スイッチ表示，シュミット・トリガ

1 A-D変換速度を上げたい

今日の要求

開発中の組み込みシステムにおいて，データ取得の部分がうまくいっていない．あれこれ調べた結果，A-D変換の速度が期待よりも少し遅いことが分かった．ソフトウェアでA-D変換の速度を上げることはできないか？

● 対策：2個使いで倍速処理

A-D変換にかかる時間は，使用するデバイス（A-Dコンバータ）で決まってしまいます．高速なA-D変換が必要な場合は，それに見合った変換時間のデバイスを選択する必要があります．

しかし，コストの関係などにより，どうしても高速のデバイスを選択できない場合，もしマイコンに複数のA-Dコンバータが搭載されていれば，簡易的に解決する方法があります．複数のA-Dコンバータに交互に処理させて，見かけ上の変換処理速度を上げるのです．

ハードウェアの構成例を**図1-1**に示します．ここでは二つのA-Dコンバータを使っています．ハードウェアとしては，単に両方のA-Dコンバータの入力をつないでおくだけです．

一方，ソフトウェアとしては，これらのA-Dコンバータを交互に動作させます（**図1-2**）．こうすることにより，理論上はデバイスの仕様（最大値）の2倍の速度でA-D変換が可能になります．

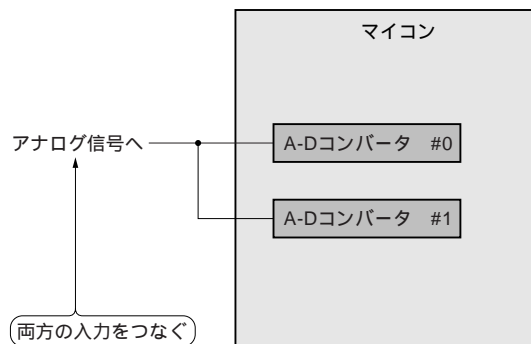



図1-1 A-Dコンバータの接続例
処理したいアナログ信号を，複数のA-Dコンバータに入力する．

● 実用上の注意点

A-D変換時間にはある程度のマージンを見ておく必要があるので，動作間隔はあまりぎりぎりに設定しない方が賢明です．また，複数のA-Dコンバータを使うことになるので，それぞれのデバイスの特性の違いが誤差として現れます．このため，一定以上の精度が必要な用途には向きません．あくまでも速度優先の場合の奇策です．

なお，デバイスの仕様上には「A-Dコンバータを何チャネル搭載」と書かれていても，実際のA-Dコンバータ（A-D変換回路）は一つで，入力だけがスイッチで切り替えられるようになっている場合があります．このような構成ではこの方法は使えません．あくまでも，A-D変換回路が複数ある場合だけです．

たち・のぶゆき
NECマイクロシステム（株）

<筆者プロフィール>
舘 伸幸．1983年入社．デバイス会社でソフトひとすじ．0x30歳へカウントダウン中．枯れ木も山の賑わいモードで活躍している．

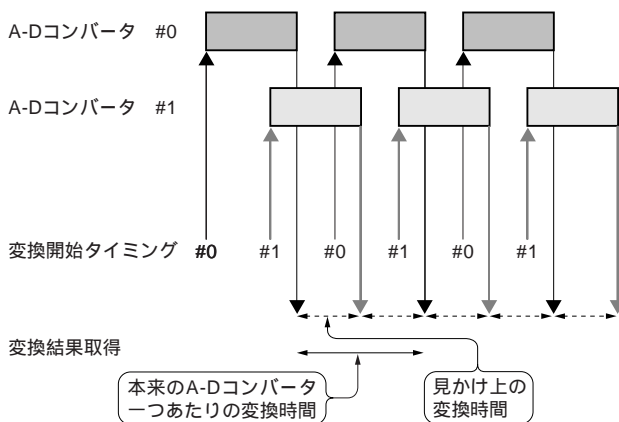



図1-2 A-Dコンバータの動作タイミング
複数のA-Dコンバータを順番に動作させることにより，A-Dコンバータ一つあたりの見かけ上の変換時間を短縮できる．

2 D-Aコンバータが足りない

今日の要求

開発中の組み込みシステムは，アナログ信号を出力する必要がある．D-Aコンバータを内蔵したマイコンは種類が少ないようだが，その中からマイコンを選ぶか，専用のD-AコンバータICを外付けする以外に，何か方法はないだろうか？

● 対策：ラダー抵抗を組むか，PWMを利用する

専用ICを外付けしないですませる一つの方法として，出力ポートに抵抗を接続して*R*-2*R* ラダー抵抗（D-A変換回路）を組むという方法があります（**図2-1**）．平易な方法ですが，多くの出力ポートを必要とするのが難点です．

別の方法として，空いている内蔵タイマを使ってPWM（pulse width modulation）出力ができるのであれば，外部にローパス・フィルタ回路を付けてアナログ電圧を作る方法があります（**図2-2**）．

PWM出力には，レジスタを二つ使用して周期とデューティの両方を変えられるものもありますが，ここではタイマをフリー・ラン動作させ，デューティのみを調整する例を紹介します．この場合，ソフトウェアの処理としては，出力したいアナログ電圧のディジタル値を出力したいタイミングでレジスタに書き込むだけですみます．タイマのフリー・ラン周期が短いほど，後段のローパス・フィルタのコンデンサ容量を小さくすることができます．また，PWMのデューティの分解能が細かいほど，アナログ出力の分解能を高くすることができます．

なお，時間変化の大きいアナログ信号の出力処理は負荷が大きいため，マイコンで処理するにはあまり向いていないことを付け加えておきます．

たち・のぶゆき
NECマイクロシステム（株）

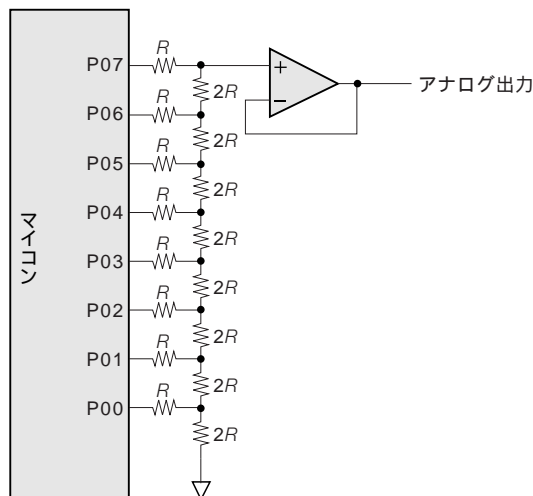



図2-1　マイコンの出力ポートにラダー抵抗を組む例
かんたんにD-A変換回路を実現できるが，多くの出力ポートが必要となるのが難点．

（**a**）一般的なPWM出力回路

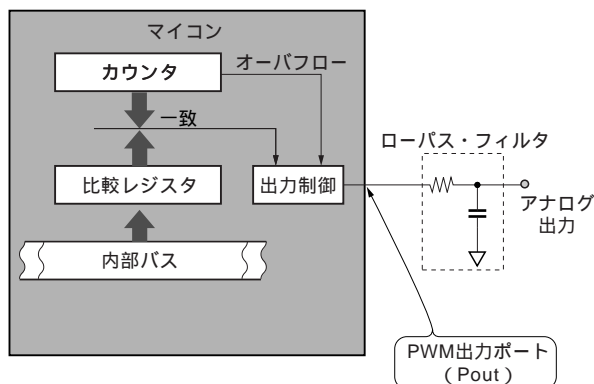
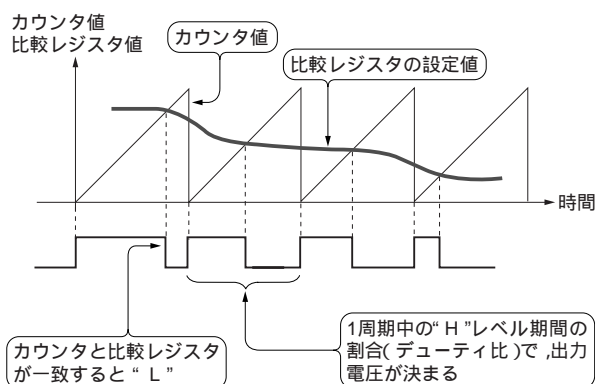
カウンタ値
比較レジスタ値
カウンタ値
比較レジスタの設定値
時間
カウンタと比較レジスタが一致すると“ L ”
1周期中の“ H ”レベル期間の割合（デューティ比）で，出力電圧が決まる

（**b**）カウンタ値と比較レジスタ値，PWM出力値の関係

図2-2　PWM出力ポートにローパス・フィルタを接続する例
タイマのカウンタが（オーバフローなどで）0になったときPoutは“ H ”になり，カウンタの値が比較レジスタ値と一致したときPoutは“ L ”になる．ローパス・フィルタからはPWM信号のデューティに比例した電圧が得られる．

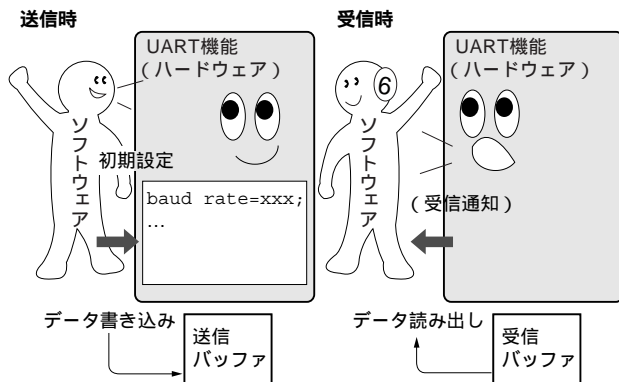
3　UARTがない

今日の要求

今回開発する組み込みシステムで，突然仕様変更が発生し，機能が追加されることになった．このためUART（universal asynchronous receiver transmitter）が必要となるが，採用したマイコンには空いているUART機能はない．コストや開発スケジュールの制約により，今から上位のマイコンに変更するわけにもいかない．なんとかUARTをソフトウェアで実現できないだろうか？

● まずはUART機能の動作を理解する

マイコンに内蔵された一般的なUART機能を用いて送信を行う場合，ソフトウェアでは通信速度などの初期設定を行った後，送信バッファにデータを書き込むだけで，あとはハードウェアが自動で出力端子から送信してくれます．受信なら，スタート・ビットの検出からデータの取り込みまではハードウェアが行ってくれるので，ソフトウェアは受信通知を受けて受信バッファのデータを読みに行くだけです．パリティ・エラーやフレーミング・エラーなどの検出もハードウェアで行うので，特定のビットを参照すれば受信状態が分かります（**図3-1**）．

UART機能をソフトウェアで実現する際には，汎用の入出力ポートを通信端子として使用します．送受信データのタイミングに合わせてポートの入出力を行う必要があるので，タイマ割り込みを使って時間を管理します．

ここで紹介する方法は，処理のタイミングが遅れると通信エラーが発生する可能性があります．割り込み処理はできるだけ高い優先順位で行う必要がありますが，それが許されるかどうかはシステム要件次第です．また，通信速度や通信頻度も，CPUの処理能力に影響を与えます．処理負荷は事前によく検討する必要があります．

● UART受信をソフトウェアで実現する

図3-2にUARTの受信処理のフォーマット（処理タイミング）を示します．スタート・ビットは，信号レベルの立ち下がりを検出することで実現できます．立ち下がりの検出は，通信に使用するポート端子がエッジ検出機能を持っていれば，それを利用するのが最適です．ない場合は，ソフトウェアでポートのレベルを監視し続ける（ポーリング）必要があります．

次に，この検出タイミングを起点として，以下の**1)**～**3)**の処理を行います．

1) スタート・ビットの確認（**図3-2**の①）

信号が“L”レベルであることを確認する．“H”レベルなら，検出したエッジはノイズと見なす．

2) データ・ビットの取り込み（8回処理する．②～⑨）

3) ストップ・ビットの確認（⑩）

信号が“H”レベルであることを確認する．“L”レベルならフレーミング・エラーとする．

①～⑩の処理のタイミングは，タイマ割り込みによって

図3-1　マイコン内蔵のUART機能があれば…

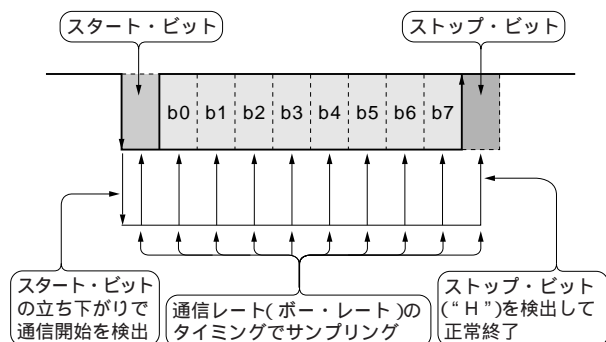



図3-2　UART受信のフォーマット

データ長8ビット，パリティなし，ストップ・ビット1ビットの例．スタート・ビットの立ち下がり検出から最初のサンプリング（スタート・ビットの確認）までは通信タイミングの半分の時間である点に注意いただきたい．

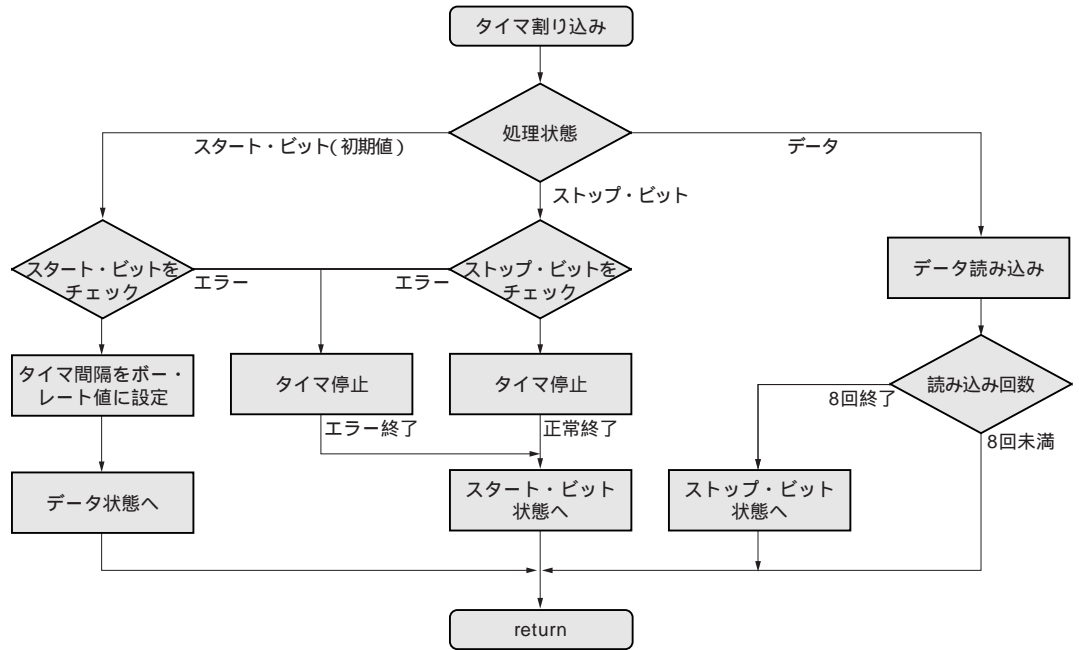



図3-3 UART受信のソフトウェア処理フロー
実使用においては，受信動作はノイズに対する考慮も必要である．

管理します．タイマ割り込みに続くUART受信の基本的な処理フローを**図3-3**に示します．

● UART送信をソフトウェアで実現する

送信の場合は，タイマを通信速度に合わせて，割り込みタイミングで次の**1)**～**3)**の処理を行います．

1) スタート・ビット"L"の出力(①)

2) データ・ビットの出力(8回処理する．②～⑨)

3) ストップ・ビット"H"の出力(⑩)

UART送信の基本的な処理フローを**図3-4**に示します．

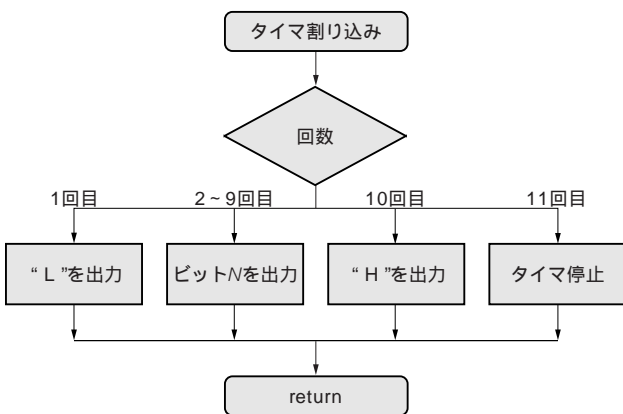



図3-4 UART送信のソフトウェア処理フロー
CPU負荷さえ許せば，送信は実用性が高い．ただし調歩同期通信なので，タイミングのずれにはシビア．処理は最優先にすることが望ましい．

● クロック同期式シリアル・インターフェースを活用する

マイコンにUARTの機能はなくても，クロック同期式シリアル・インターフェースならある，という場合があります．この場合，前述の処理手順のうち，データの送受信部分(スタート・ビットとストップ・ビット以外)にクロック同期式シリアル・インターフェース機能を使うことで，ソフトウェアの負荷を減らすことが可能です．ただし，クロック同期式シリアル・インターフェースの通信端子のレベルをソフトウェアで操作できるようになっていることが条件となります．

また，入出力ポートとの兼用端子になっているような場合，シリアル・インターフェース・モードとポート・モードの切り替え時にノイズが出力されることがあります．デバイスの制限事項などをよく調べてから利用しましょう．

たち・のぶゆき
NECマイクロシステム(株)

2

4 UARTの制御信号がない

今日の要求

開発中の組み込みシステムに外部モジュール(ハードウェア)が追加されることになった．外部モジュールとのインターフェースはUARTである．マイコンのUARTチャネルも余っていたので問題はなさそうだ．ところがよく見ると，外部モジュールのUARTには，データ線以外にRTS，CTSと記述された2本の信号がある．どうやら，この外部モジュールと通信するには，ハードウェアによるフロー制御が必要なようだ．マイコンのUARTにはデータ線しかないが，どう実装すればよいだろうか？

● 対策：空きポートを使って制御信号を作成

UART通信には，RTSとCTS(または，DTRとDSRなど)というハードウェア・フロー制御信号(ハンドシェイク線)を使う場合があります(**図4-1**)．一方，マイコン内蔵のUART機能はRXDやTXD(データ線)のみの場合があります．

通信相手がRTSとCTS信号を必要とする場合は，2ビットの汎用I/Oポートを使って，ソフトウェアでRTS/CTS信号を制御することによって対応できます．ただし，ハードウェアの支援は一切受けることができません．すべてをソフトウェアで制御する必要があります．

自分(マイコン)が受信できないとき(例えば受信バッファがあふれそうなとき)は，RTSをインアクティブのレベルに設定し，相手側にこれ以上受信できないことを示します(**図4-2**)．送信するときはCTS端子をチェックし，アクティブならば送信開始，インアクティブならば送信を保留します(**図4-3**)．

基本動作としてはたったこれだけですが，フロー制御をソフトウェアで実装する場合，以下の点に注意する必要があります．

1) RTSをインアクティブに設定してから何s(秒)で相手側が送信を停止するのか

RTSをインアクティブに設定した時，すぐに通信相手が送信を停止してくれるとは限りません．相手がRTSのインアクティブを認識するまでの間は受信可能な状態を維持し，

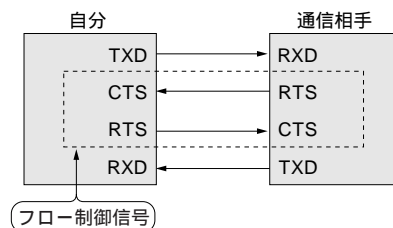



図4-1 UARTのフロー制御信号

UARTのフロー制御は，RTSとCTSなどのハードウェア信号を使う場合と，XonとXoffなどのソフトウェア信号を使う場合がある．

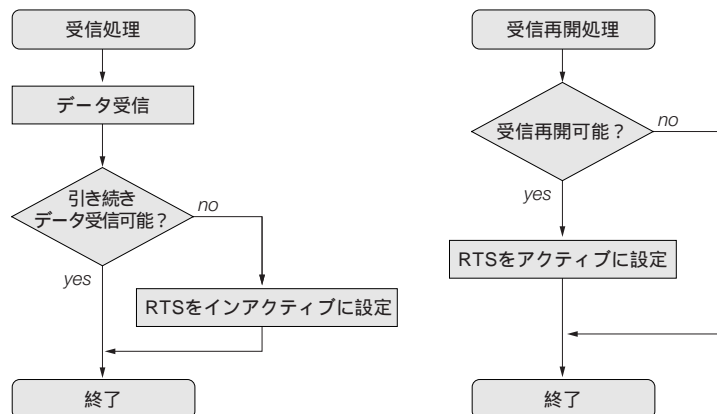



図4-2 受信時のフロー制御

処理自体はシンプルである．受信可能かどうかの判断には，通信相手の仕様も含めて検討する必要がある．

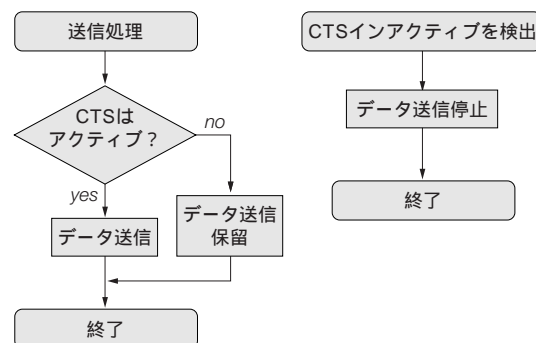
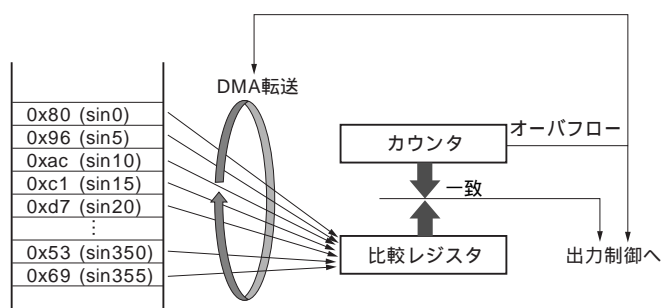



図4-3 送信時のフロー制御

DMA(direct memory access)などを使って連続送信する場合は，CTSを割り込み信号として扱うなどして，CTSインアクティブを検出する必要がある．1バイト送信ごとにCTSを確認する場合は，右のCTSインアクティブの検出は不要．

コラム
意外と便利なDMA

DMA(direct memory access)は，工夫すると意外と便利なマイコンの周辺機能の一つです．例として，「2．D-Aコンバータが足りない」で紹介したPWM出力における使用例を示します．

図Aは，PWM出力のデューティを変更することによって正弦波信号を生成しています．レジスタの設定値(PWM信号のデューティ)をソフトウェアで定期的に書き換えるのは負荷が大きいのですが，DMA(direct memory access)転送をうまく使うことにより，負荷をかけずに繰り返し波形を出力できます．

図A　DMA転送を利用して正弦波信号を出力する
DMAは，工夫すると意外と便利な周辺機能の一つである．

たち　のぶゆき
NECマイクロシステム(株)

送られたデータを破棄せずに保存するためのバッファ領域が必要です．

2) RTSをインアクティブからアクティブに戻したとき，相手はどこからデータ送信を再開するのか

受信バッファに余裕ができたらRTSをアクティブに戻して受信を再開するわけですが，通信相手はデータのどこから送信を再開してくれるのでしょうか．中断したデータの途中から？ それともフレームの最初から？ 相手の仕様を確認しておく必要があります．

3) データ送信中にCTSがインアクティブになったら，何s以内に送信を停止しなければならないのか

これは**1)**と逆のパターンです．データ送信中に通信相手がデータを受信できなくなり，CTSがインアクティブになったとき，一定時間(相手がデータ受信できる時間，またはバイト数)以内に送信を停止する必要があります．

4) データ送信中にCTSがインアクティブになったら，どのように送信を停止するべきか

CTSがインアクティブになったからといってUARTの送信動作を突然停止すると，CPUによっては1バイトの中途半端な位置でデータが切れてしまうことがあります．この場合，通信相手側でフレーミング・エラーが発生する可能性があります．これを回避するためには，バイト単位で送信を停止する工夫が必要です．最も簡単な方法は，1バイト送信ごとにCTSを確認することです．

5) CTSがインアクティブからアクティブに戻ったとき，どこからデータ送信を再開するべきか

これも**2)**と逆のパターンです．CTSがインアクティブからアクティブになり，データ送信を再開する場合，データのどこから送信を再開するべきなのかを確認する必要があります．

なの・ひびき
NECマイクロシステム(株)

<筆者プロフィール>
名野 響．ものづくり大好き人間．最近，ドロドロした組み込みソフトウェア開発が快感になってきました．

2

5 ポートが足りない

今日の要求

開発中のシステムにおいて，利用可能な汎用I/Oポート数が足りないことが分かった．マイコン選定時には十分なポート数だと判断したのだが，多くのピンが汎用I/Oポートと内蔵機能の兼用として割り当てられており，使えるポート数が減ってしまったのだ．今からマイコンを変更することは許されない．どうにかして，使用するポートの数を減らせないだろうか？

```
fのLEDだけを点灯するとき
  P0 = 0x29;
  /* P00 : 1  P03 : 1
     P01 : 0  P04 : 0
     P02 : 0  P05 : 1 */

aとiのLEDを点灯するとき
  P0 = 0x34;
  /* P00 : 0  P03 : 0
     P01 : 0  P04 : 1
     P02 : 1  P05 : 1 */
  P0 = 0x19;
   /*P00 : 1  P03 : 1
     P01 : 0  P04 : 1
     P02 : 0  P05 : 0 */
を交互に繰り返す
```

図5-1　LEDをマトリックス接続する

P00-P02を列，P03-P05を行として制御する．ソフトウェアで点灯させたいLEDの列をONにしながら，点灯させたくないLEDの行をONにすることにより，ねらったLEDが点灯する．複数のLEDを点灯させたい場合は，各LEDを交互に点灯させることにより，両方点灯しているように見せられる．

● 対策その１：マトリックス回路にする

例えば，9個のLEDの出力がある場合，各LEDにポートを1本ずつ割り当てると9本のポートが必要となります．これを，3×3のマトリクス回路でダイナミック表示すれば，6本のポートで実現できます（**図5-1**）．

同様に，9個のスイッチ入力がある場合も，3×3のマトリックス・スイッチで入力すれば3本の入力ポートと3本の出力ポート（合計6本）で実現できます．ソフトウェアではダイナミック表示のための処理が必要となります．

● 対策その２：A-Dコンバータで入力する

マイコンの動作モードや製品の機種情報などの設定（例えば16種類の設定）に汎用ポートを使う場合，入力ポート4本を使う（2進数で表現すると“0000”～“1111”）のが一般的です．これを，1チャネルのA-Dコンバータを使うことで，1本のアナログ入力で同等の機能を実現できます．

外付け抵抗を使ってアナログ電圧を入力する回路を**図5-2**に示します．また，8ビットA-Dコンバータを使った場合の入力電圧の割り当て例を**図5-3**に示します．

この方法は将来の機能拡張にも簡単に対応できる利点があります（例えば，設定数が追加された場合もアナログ電圧と設定値の対応を変更すればよい）．ただし，設計にあたってはA-Dコンバータの保証精度やノイズの問題を考慮する必要があります．ソフトウェア処理としては，値を複数回取得して精度を上げるなどの工夫が必要です．

図5-2　汎用ポートをアナログ入力端子で代替する

動作モードなどの設定値をアナログ値（電圧）として入力する．アナログ値の設定には，ロータリ・スイッチなどを使用する．この方法を採る場合は，ノイズを考慮して余裕をもってタイミングを見積もる必要がある．

上位4ビットが設定モードの値と同じになるように回路を設計すると，ソフトウェア処理が楽になる

設定値	入力電圧	A-D変換値
0	0	0x00
1	0.47	0x18
2	0.78	0x28
3	1.09	0x38
⋮	⋮	⋮
14	4.53	0xe8
15	5	0xff

図5-3　設定値とA-D変換値の例

ここでは，ソフトウェアでこの端子のA-D変換結果から上位4ビットを参照することにより，設定値を読み込めるようになっている．このように，ハードウェアで解決する場合も，常にソフトウェアの負荷軽減を考慮するとよい．

たち・のぶゆき
NECマイクロシステム（株）

6 タイマが足りない

今日の要求

開発中の組み込みシステムで，タイマの数が足りないことが分かった．今回のソフトウェアでは，一定間隔の定期的な処理やハードウェアの処理を待つためのウェイトなどが数多くあり，マイコンに用意されているタイマ機能では数が足りなかったのだ．

● 対策：一つのタイマで複数のタイマの役割を果たす

このような場合でも，ソフトウェアによって疑似的に複数のタイマが同時に動作しているように見せかけることができます．例えば，一つのハードウェア・タイマを利用して，50msインターバル処理と70msインターバル処理という二つのタイマ機能を同時に担当させることを考えてみます．

リスト6-1にサンプル・プログラムを示します．①のhard_timer_10ms関数は，ハードウェア・タイマによって10msごとにDONEを返却します．そこで，50msと70msのタイマ用カウンタを10msごとにカウントアップし，②と③で各カウンタが満了している場合に所望の処理を実施させる仕組みです．

この方法を採れば，理論上はいくつでもタイマを増やせます[注6-1]．ただし，各タイマの処理時間の合計（whileループを1回実行するのにかかる処理時間）を，少なくとも10ms以内とする必要があります．10msを超えてしまう場合，ハードウェアで10msごとにカウントしているはずが，カウントアップを飛ばしてしまうことになり，正確な時間で処理できなくなります．

そのような場合は，基準タイマの分解能を10ms以上とするか，カウントアップをタイマの割り込み処理で行うなど，さらに一工夫することで対応できるでしょう．この方法は，OSを利用しないような比較的規模の小さなソフトウェアで有効な手段です．

リスト6-1 一つのタイマで複数のインターバル処理を担当する

```
 int Count50ms = 0;
 int Count70ms = 0;

hard_timer_start(); /* 10msタイマ・スタート */

while( 1 )
{
    if ( hard_timer_10ms( ) == DONE )   ……①
    {
        Count50ms++;
        Count70ms++;
       if ( 5 <= Count50ms )
        {
            Count50ms = 0;
            /* 50msインターバル */        ……②
        }

       if ( 7 <= Count70ms )
        {
            Count70ms = 0;
            /* 70msインターバル */        ……③
        }
    }
}
```

注6-1：実際は，カウンタ用変数のための領域を確保する必要があるので，無限に増やせるわけではない．

きのした・ひであき
NECマイクロシステム（株）

<筆者プロフィール>
木下秀昭．入社以来，マイコンや車載LAN関係の組み込みソフトウェアなどを担当．いつの間にやら職場では自分より若い人も多くなったが，まだまだ修行の毎日．

7 チャタリングを回避する

今日の要求

現在開発中の製品にはスイッチが付いている．ハードウェア部門から試作ボードを受け取ったとき，「分かっていると思うけど，チャタリングの処理はソフトウェアでよろしくね」と釘を刺された．とりあえず「はい」と答えたが，どうすればよいのかはこっそり先輩に聞いてみよう….

● スイッチの幻想（？）と現実

マイコンに対する入力として，I/Oポートなどにスイッチを接続することがあります．スイッチは，操作した瞬間にピタリと接点がくっつくイメージがありますが，実際は細かくON/OFFが繰り返されてから値が安定します（**図7-1**）．

このような現象をチャタリング（chattering）と言います．リレーの接点などではチャタリングが数十ms継続する場合もありますし，スライド・スイッチではスイッチ・ノブを動作させている間中，チャタリングが発生します．

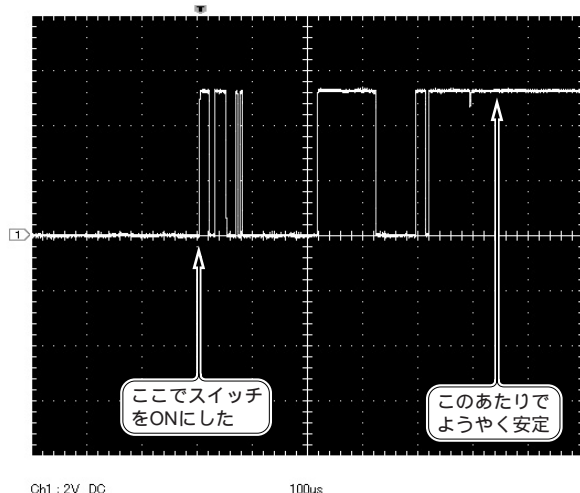



図7-1　チャタリングの観測例

実際に小型のトグル・スイッチで観測した例．横軸の1目盛りは100μsであり，ここではおよそ500μsにわたってON/OFFが繰り返されている．ほとんどの機械式スイッチにおいて，このような現象が見られる．

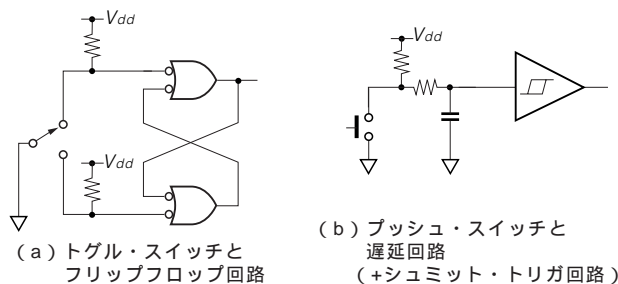



図7-2　ハードウェアによる対策例

（a）は，トグル・スイッチ（2接点型）の場合に，フリップフロップ回路を使った例．フリップフロップ回路が最初に入力された値を保持するため，チャタリングを除去できる．（b）は，プッシュ・スイッチの場合に，遅延回路とシュミット・トリガ回路を使った例．遅延回路でなまった波形をシュミット・トリガで整形する．

● 対策：ソフトウェアで値を判定

チャタリングは，ハードウェアで対策する方法もありますが（**図7-2**），部品点数が増えるのでコスト面で不利になるという問題点があり，通常はソフトウェアで対応します．

最も単純な方法を**図7-3**に示します．スイッチONを検出したら，間隔を置きながら値を数回読み込みます．**図7-3**では3回読み込んで，3回ともONであれば「スイッチはONである」と判断します．この方法は単純ですが，スイッチの再読み込みの間，処理が占有されてしまうという欠点

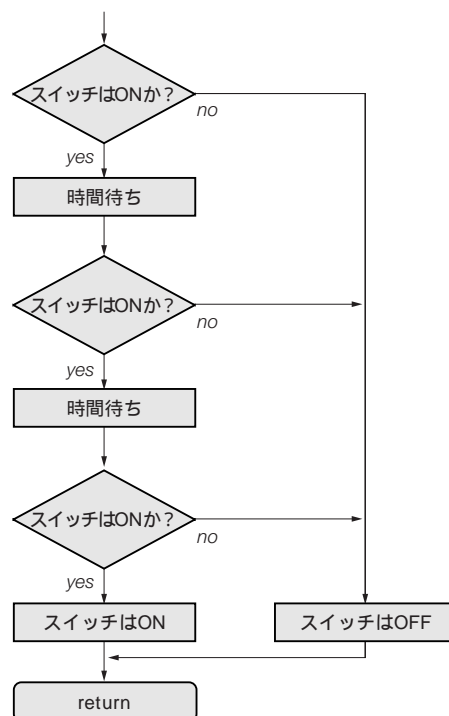



図7-3　単純なチャタリング対策例

値を複数回読み込み，安定してONになったタイミングを検出する．

図7-5
仮想スイッチとアプリケーション・ソフトウェアの動作タイミング例
サンプリング周期はチャタリング発生時間よりも長くとる必要がある．逆に，サンプリング周期よりも短時間のスイッチ操作は検出できないこともあるので注意．

があります．

一方，**図7-4**のように，ソフトウェアで仮想スイッチ・モジュールを持つ方法もあります．この仮想スイッチ・モジュールは一定時間ごとに実際のスイッチの状態をチェックして，スイッチの状態を保存しています．アプリケーション・ソフトウェアからの問い合わせには保存されている現在の状態を返すため，アプリケーション・ソフトウェアの処理時間を占有しません．

この動作タイミング例を**図7-5**に示します．サンプリング周期をチャタリング発生時間よりも長く取ることにより，チャタリングによってスイッチ状態が短時間に不安定になることを回避しています．ただし，サンプリング周期をあまり長くするとスイッチ操作を検出できない場合が出てきてしまうので，システムに応じた設計が必要です．

図7-4　仮想スイッチを利用したチャタリング対策例
スイッチを仮想化することで移植性も向上する．

たち・のぶゆき
NEC マイクロシステム（株）

8 スイッチによるカウントアップをいかに実装するか

今日の要求

開発中のシステムに，ちょっとしたボリューム増減の入力機構を作りたい．1回操作するごとに確実に1個ずつカウントアップしてくれる快適さが欲しい．また，一気に増減するような操作も実現したい．

● プッシュ・スイッチは「押しすぎ」に注意

まずはプッシュ・スイッチを2個使用することを考えてみましょう．個人的には「カチッ」と音がするスイッチが好みです．そして，「カチッ」と1回押したら確実に一つ，「カチカチカチカチカチッ」（その間0.5秒）と5回押したら確実に五つカウントアップするのが好ましい操作感です．どうすればこの操作感が得られるでしょうか．

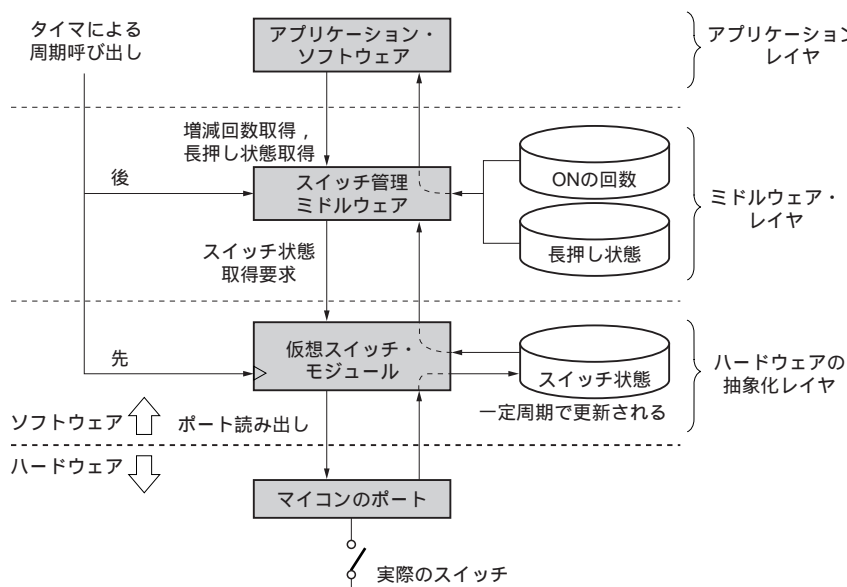
人間の最大連打能力（？）を1秒間に20回と仮定すると，その周期は50msとなります．使用するスイッチは，このON/OFFを確実に拾える程度にチャタリング期間が短いものを選ぶ必要があります．また，増減それぞれに1個のスイッチを割り当てるので，合計2本のポートを使用できるマイコンが必要です．

ここでは，ソフトウェアでチャタリングを処理するものとして考えてみます．スイッチの値をソフトウェア側で保存しておく仮想スイッチ・モジュール[注8-1]を用意した場合を考えてみましょう．

ここで，仮想スイッチ・モジュールが行ってくれるのは「ON/OFF状態の識別」だけであることに注意が必要です．もしアプリケーションのポーリング処理の中で，

```
if ( 仮想スイッチがONならば )
    ボリュームアップ;
```

のようにプログラミングしてしまうと，人間からしてみれば1回のスイッチ押下でも，このif文が何度も実行されてしまい，「ドドドド」とボリュームが上がってしまいます．これでは「カチッで1回」の操作感は得られません．音量なら耳が痛くなり，輝度なら突然まぶしくなってしまうかもしれません．

注8-1：前項「7．チャタリングを回避する」を参照．

（a）モジュール構造

モジュール名	責 務
アプリケーション・ソフトウェア	押された回数や長押し状態に応じて制御対象の値を増減する
スイッチ管理ミドルウェア	押された回数や長押し状態を管理/提供する
仮想スイッチ・モジュール	チャタリングの影響を除去し，スイッチのON/OFF状態を管理/提供する

（b）モジュールの責務分担

図8-1 スイッチ管理ミドルウェアで仮想スイッチ・モジュールからスイッチ状態を取得する

モジュール構造を設計するとき，レイヤ（層）の上下は「どちらがハードウェアに近い存在か」でおのずと決まる．それに対して，スレッド割り付けや呼び出し順序は，リアルタイム要求を考慮し，設計して決めている．ここで，レイヤの異なる「仮想スイッチ・モジュール」と「スイッチ管理ミドルウェア」を同じタイマ・スレッドで呼び出すのは，仮想スイッチの状態変化を取りこぼさないためである．

そこで，以下の内部機能を実装し，押した回数をきちんとカウントする必要があります．

- ON → OFF や OFF → ON の変化を検出する
- ON → OFF や OFF → ON の変化の回数を数える
- 長押し開始と長押し終了を検出する

これらを実現するための設計案はいくつか考えられます．ここでは，アプリケーションと仮想スイッチ・モジュールの間にミドルウェアを設けて，ミドルウェアが周期的に仮想スイッチ・モジュールをポーリングするということにしましょう（**図8-1**）．

今回の要件では，ON/OFF の変化を確実に拾いたいので，イベント検出や長押し開始/終了の検出は，仮想スイッチ・モジュールのスイッチ状態（ON/OFF）の識別と同じ周期で行う設計を考えてみます．このようにしておくと，

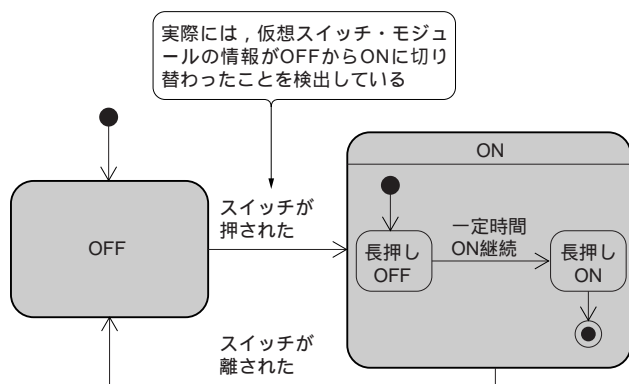



図8-2　スイッチ管理ミドルウェアの状態モデル

スイッチ管理ミドルウェアはOFF状態から始まる．スイッチが押されたことを検出するとON状態に遷移し，一定時間ON状態が継続すると，長押し状態もONであることを認識する．図に表さないと難しく感じるかもしれないが，こうして状態図に表してみると，それほど複雑ではないことが分かる．

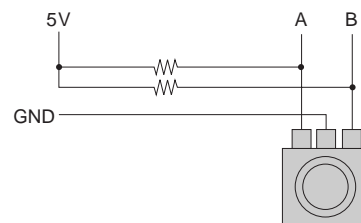
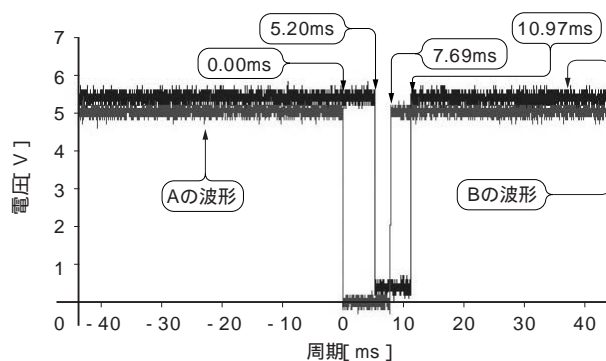



図8-3　ロータリ・エンコーダの波形

今回筆者が試したロータリ・エンコーダは，チャタリングの少ないものだった．

（**a**）モジュール構造

モジュール名	責　務
アプリケーションソフトウェア	カウントアップ/ダウン回数に応じて制御対象の値を増減する
ロータリ・エンコーダ・モジュール	カウントアップ/ダウン回数を管理/提供する

（**b**）モジュールの責務分担

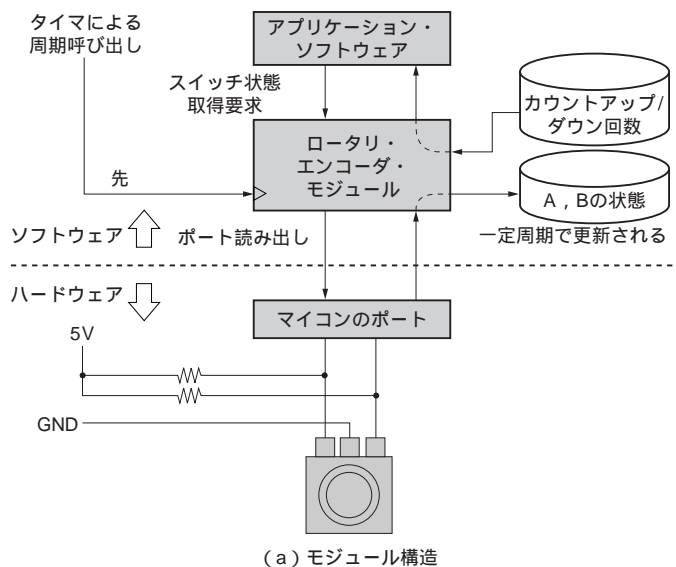



図8-4　ロータリ・エンコーダ・モジュールで回転方向を検出する

ロータリ・エンコーダ・モジュールで回転方向を検出し，カウントアップ/カウントダウン回数を管理する．ここで，図8-1（b）において「押された回数」というのは，本質的には「カウントアップ/カウントダウン回数」であるのかもしれない，ということに気づく．

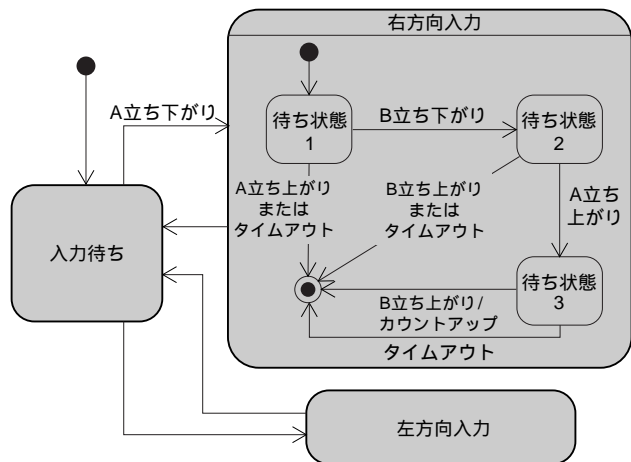



図8-5　ロータリ・エンコーダ・モジュールの状態モデル
AとBの立ち上がり/立ち下がりが所定の順序で発生した場合のみカウントアップする．それ以外の順序で入力が発生した場合にはすべてエラーとしていったん入力待ちに戻る．なお，左方向入力状態については表記を省略した．

上位アプリケーション・ソフトウェアにて「スイッチが押されている間ずっと行う処理」と「スイッチが押された回数分だけ行う処理」の両方を簡単に実装できるようになります（**図8-2**）．

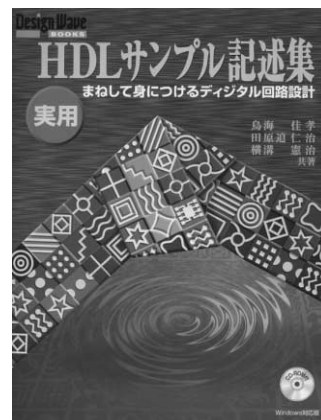
● ロータリ・エンコーダによる設計解

さて，I/Oポートを2本使用するのであれば，スイッチではなくロータリ・エンコーダを使用する手もあります．ボリュームを一気に増減したいときにも，ロータリ・エンコーダならグルンと回せばよいので操作が直感的です．ソフトウェアとしても「長押し処理」の検出が不要なため，より簡単に実装できます．また，カチカチという操作感のあるロータリ・エンコーダは，筆者の個人的な好みにも合います．

まずロータリ・エンコーダの回路構成と波形の例を見てみましょう．ロータリ・エンコーダを「カチッ」と1回だけ回すと，端子A，Bから**図8-3**のような波形が出ます．Aが先に立ち下がるかBが先に立ち下がるかは回す方向によって異なります．このA，Bを，マイコンのI/Oポートに接続し，ポーリングによって増減の回数を数えることを考えてみます．

波形と波形の間は1ms程度の余裕があるので，波形のポーリングは100μ～200μsでよいでしょう．そして，A端子，B端子の立ち下がりや立ち上がりをイベントとして取り扱う状態モデルを構築すれば，回転の方向を容易に検出できます．この考えに基づく設計を紹介します（**図8-4**，**図8-5**）．

もり・たかお
三栄ハイテックス（株）ソフト開発部

<筆者プロフィール>
森 孝夫．組み込みシステムやソフトウェアのモデリング・設計・検証関連のコンサルティング，火消しを仕事としている．趣味はサッカー．なぜか今年は出場したフルコート・ゲーム，フットサルの全てで得点を挙げている．最近感動したものは「のだめカンタービレ」，吾妻橋アサヒビールタワーの黒ビール，そしてETロボコン東海地区の皆様の走りです．

9 ハードウェアの動作完了を待つ方法

今日の要求

開発システムの詳細設計書を見ると，「ここでソフトウェアはハードウェアの処理を待つ」と書いてある．さて，待つ処理はどのように実現すればよいだろうか？

● 周辺回路の回復（I/O リカバリ・タイム）を待つ場合

CPU に比べて動作の遅い周辺回路に対して連続してアクセスする場合，少し待たないと次のアクセスを受け付けられないことがあります．回路側でWAIT信号などを使い，自動的にCPUを待たせる構成になっていれば問題ありませんが，そうでない場合はソフトウェア側で対応する必要があります．どのように時間待ちをするかは，周辺回路の仕様によります．

1）クロック数で待ち時間が規定されている場合

必要なクロック数に相当する命令を記述します．一般にはnop命令を使います．C言語では記述できないので，アセンブリ言語による記述が必須となります．

2）数ms以上（msオーダ）の待ち時間

CPUの動作速度に依存しないように，ハードウェア・タイマを使って待ち時間をかせぐのが正しい方法です．

3）1ms未満（μsオーダ）の待ち時間

μsオーダの時間待ちにハードウェア・タイマを使うと，設定のオーバヘッドだけで待ち時間をオーバしてしまうこともあります．そこで，やむを得ずソフトウェア・タイマ（単純なループの回数で待ち時間を稼ぐ）を使います．しかし，ソフトウェア・タイマはCPUの動作速度に依存するので，移植やバージョンアップ時に不具合要因となります．予防措置として，速度依存部分を明示するような実装の工夫をしておく必要があります[注9-1]．

● 処理終了を待つ場合

A-D変換のように，変換終了を待たないと正しい結果を読み出せないような処理の場合，処理完了で割り込みを発生させるか，あるいは処理完了フラグを立て，ソフトウェア側で読んで確認する方法をとります（**リスト9-1**）．

ここで注意が必要なのが，完了フラグ待ちループです．ハードウェア（ここではA-Dコンバータ）が故障して完了フラグが立たなくなってしまうと，ここで無限ループとなってしまいます．

「ウォッチドッグ・タイマを使っていればリセットがかかるから，システムがハングアップすることはない」と思いがちですが，リセットしても故障ハードウェアが復活しなかった場合，無限ループ→リセットを繰り返してしまい，見た目上，ハングアップと同様の症状に陥ります．

これを防ぐためには，ハードウェアの処理完了待ちループに回数制限条件を加えるといった，故障検出手段を講じておく必要があります（**リスト9-2**）．

たち・のぶゆき
NECマイクロシステム（株）

注9-1：例えば，ループ回数を記述した「環境依存ヘッダ・ファイル」を作るなどの方法がある．

リスト9-1　A-Dコンバータの処理完了を待つプログラム・コード（一部）

```
reg_u1AD = 1;                    /* A-D変換開始 */
while ( (reg_u1ADM & 0x01) == 0x00 ){
     ;
}                                /* 変換完了待ち */
```

リスト9-2　ハードウェアの処理完了待ちループに回数制限条件を加えたプログラム・コード（一部）

```
if ( u1sAdCond == OK ){
/* 故障が検出されたら，次からはエラーを返すだけの処理にする */

  u1tLoop = AD_LOOP_LIMIT;

  reg_u1AD = 1;            /* A-D変換開始 */
  while (     ( reg_u1ADM & 0x01) == 0x00 )
         && ( 0 < s2tLoop )  ){

     s2tLoop--;
  }                        /* 変換完了待ち */

  if ( s2tLoop <= 0 ){
     u1sAdCond = FAIL;
     u2tResult = ERROR;
  }
  else{
     u2tResult = (u2)reg_u1ADR;
  }
}
else{
  u2tresult = ERROR;
}

return u2tResult;
```

静的変数u1sAdCondには初期化処理でOKを設定しておく

10 誤動作時にリセットがかかるようにするには

今日の要求

開発中の組み込みシステムは，「万が一誤動作してもリセット動作するように設計してくれ」と言われている．そう言えば，割り込みベクタ・テーブルのうち，使っていないものがあったなぁ…．もし，ノイズなどの誤動作でそこを参照してしまったら，システムはどういう動きをするのだろうか？

● 未使用のベクタ・テーブルを放置しない

マイコンには，割り込みに対応してその処理モジュールへ飛ぶためのテーブルが用意されています．これをベクタ・テーブルと言います．マイコンが持つすべての割り込みを使用することはまれです．ここで，使っていないベクタ・テーブルを放置しておくと，トラブルが発生したときに思いもよらない動作に発展してしまうことがあります．

そこで，未使用のベクタ・テーブルはすべて同じ番地を参照するようにしておきます．一方，参照先のアドレスには無限ループを記述しておきます(**図10-1**)．このようにしておくことで，誤動作により本来あり得ないはずのベクタ・テーブル参照が発生した場合にも，必ず，安全な無限ループが実行されます．その結果，ウォッチドッグ・タイマが動作してシステムがリセットされ，正常動作に戻る可能性が高まります．

図10-1　空きベクタ処理

未使用のベクタ・テーブルはすべて同じ番地を参照するようにしておく．また，似たような方法として，例外命令(ソフトウェア割り込みなど)で埋めておく方法もある．

● 無限ループを勧める理由

「無限ループ→ウォッチドッグ・タイマによるリセット」に持ち込むのではなく，状況を判断してソフトウェア的に復帰動作をさせたいと思うかもしれません．しかし，設計外の割り込み動作が起こっている時点で，もはやシステムには重大な異常が発生していると考えるべきです．下手に復帰動作を画策すると，かえって傷を深くしてしまう可能性もあります．ここは素直にハードウェア・リセットに持ち込むのが最善策でしょう．

また，「無限ループではなくリセット・ベクタに飛ばしても同じではないか」と思われるかもしれませんが，その場合，内蔵周辺機能はリセットされません．

たち・のぶゆき
NEC マイクロシステム(株)

11 出力ポートの設定で気を付けたいこと

今日の要求

抵抗でプルアップされたアクティブ“L”(通常は“H”レベル)のポートを，出力ポートとして使おうとしている(**図11-1**)．初期化の手順として，まずポート・モードを「出力」に設定し，次にポートの値を“H”に設定した．すると，ポートに接続されている装置が誤動作してしまった．いったい，なぜ？

● レジスタの設定順序に気を付けよう

一般的に，I/Oポートは以下のような設定レジスタを持っています[1]．

1) モード・レジスタ

指定したI/Oポートを「入力」モードにするか「出力」モードにするかを設定します．この例では，リセット時の初期値は「入力」モード“H”レベルになっていました．

2) ポート・レジスタ

実際にI/Oポートを使って入出力する値を設定します．この例では，リセット時の初期値は“L”レベルになっていました．

図11-1のPxx端子は抵抗でプルアップしているので，電源を入れた直後は“H”レベルで安定しています．ここで，初期化処理により，I/Oポートのモード・レジスタを「出力」に設定するとどうなるでしょうか．この瞬間，つまり使用するI/Oポートのモードを「出力」に設定してから，ポート・レジスタに正しい値を設定するまでの間，ポートには，“L”レベルが出力されてしまいます(**図11-2**)．このようなパルスは接続先にとって何らかのトリガ信号と見なされる可能性があり，誤動作を誘発します．

● ポート・レジスタを設定する

この件についての対策は，マイコンの仕様によって異なりますが，簡単です．ポート・モード・レジスタを出力に切り替える前に，ポート・レジスタに本来出力するべき初期値(今回の例では，“H”レベル)を設定しておけばよいだけです．

参考・引用*文献

(1) * V850ES/JG2のユーザーズ・マニュアル(資料番号：U17715JJ2V0UD00)，http://www.necel.com/nesdis/image/U17715JJ2V0UD00.pdf

たち・のぶゆき
NECマイクロシステム(株)

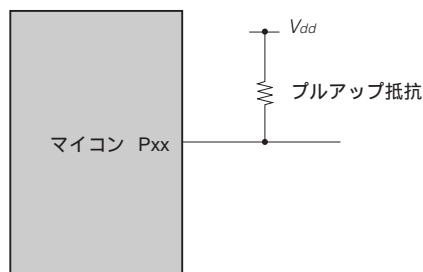



図11-1 アクティブ“L”の出力ポート
このI/OポートPxxにはプルアップ抵抗が付いているので，マイコンの電源を入れた直後の端子の電圧レベルは“H”で安定している．

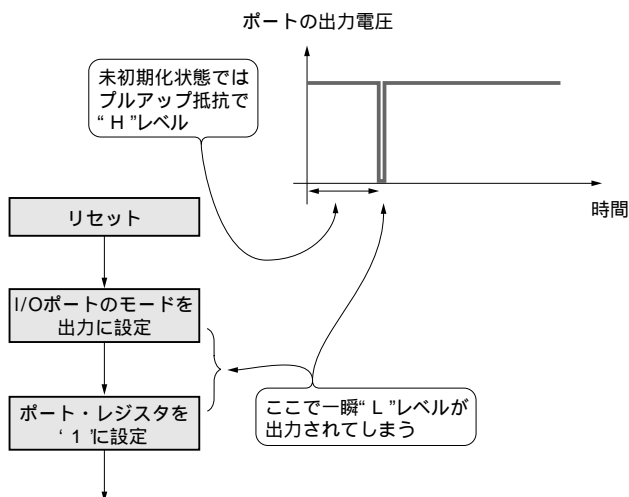



図11-2 I/Oポートのレジスタの設定順による誤動作
ポートの接続先が信号のエッジをトリガとして動作する場合，誤動作を引き起こしてしまう．

コラム

マイコンによる組み込みシステム制御，転ばぬ先の杖

マイコンによる組み込みシステム制御は，以下のような点に日常的に注意しておくことにより，トラブルを未然に防ぐことができます．

● 注意その１：兼用端子に注意する

一般にマイコンは，汎用性を高めるために可能な限りの周辺機能を搭載しています．しかし一方で，限られたピン数のパッケージに納めるために，同じ端子に複数の機能が割り当てられています．単純に「UARTが2チャネル，外部入力付きタイマが一つ，ポートが20本」という条件で選んだら，実はUARTのTx端子とタイマの外部トリガ端子，さらにはポートのどれかが一つの兼用端子に割り振られていた，という場合もあり得ます．

A-Dコンバータのチャネル数についても同じことが言えます．例えば「8チャネル」というスペックの場合，たいていA-D変換回路は一つで，それに入力の切り替え機能が付いている構成です．要求仕様として複数のA-Dコンバータを同時に動作させることが求められている場合，これではうまくいきません．

マイコンを選定する際には，カタログに掲載されているスペック（機能一覧）だけでなく，兼用端子や内部ブロック図もよく吟味して，本当に使いたい機能が使えるのかどうかを確認しましょう．

● 注意その２：最新の制限事項を確認する

使用するマイコンが決まったら，まず制限事項に関する最新情報を入手して確認することが大切です．マイコンにも不具合や制限事項があることがあります．これを知らずにプログラムを作ると，プログラムは正しいはずなのに正常に動作しないというトラブルに遭遇します．この場合，問題はデバイスの内部にあるので，いくらソフトウェアを見直しても解決せず，時間を浪費することになります．

マイコンの情報入手と同時に，使用する開発ツールについても不具合情報を調べておきましょう．

なお，開発完了後も，使用したデバイスに関する情報には十分注意しておくことが大切です．制限事項はバージョンの更新時に修正される場合があります．これにより，制限事項を回避するための施策が逆に不具合を引き起こしてしまう可能性もあるのです．また，開発時には知られていなかった制限事項が後から見つかることもあります．

● 注意その３：初期化時は全レジスタを記述する

ハードウェアの初期化処理漏れによるトラブルはやっかいな場合が多いです．ある特定の条件[注1]でのみ発生するような不安定な動作として出現することがあるためです．対象のシステムで使用する機能（のレジスタ）だけを記述していると，思わぬ設定漏れを作り込みやすくなります．

「UARTを使うだけなのに，実はその割り込み信号は途中に許可スイッチがあって，それは使用予定のないポート機能の中にあった」といった例は少なくありません．この場合は明らかに動作しないので多少の工数を失うだけですが，場合によっては不安定な状態を引き起こす組み合わせもあります．

対象システムでの使用の有無に関わらず，すべてのレジスタを列挙し，レジスタの初期値に頼ることなく初期化するようにしましょう．この場合，デバイスのマニュアルに対応した順序（たいていはアドレス順）に記述しておくことで，レビューでチェックしやすくできます．

リストA　レジスタINFREGが0でなくなるまで待つループ処理

```
#define        INFREG  (*(unsigned char
*)0xffff1234)

void func()
{
      INFREG = 0;
           :
         (中略)
           :
      while(INFREG == 0){
                ;
      }

      return;
}
```

● 注意その４：volatileに注意する

C言語で，マイコンの内蔵レジスタ設定を自分で記述する場合によく失敗するのが，変数のvolatile宣言忘れです．

volatile宣言はコンパイラへの最適化抑制命令です．例えば，特定のレジスタが0x00でなくなるまで待つループを考えましょう（**リストA**）．このレジスタは0xFFFF1234番地にあって，何らかのハードウェア条件[注2]で値が書き換わるものとします．

コンパイラから見ると，このソースにおいて`INFREG`が0以外に変化することはあり得ないので，コンパイル結果として単なる無限ループ処理に置き換えてしまうことがあります．これはレジスタに限ったことではなく，例えば割り込みハンドラで書き換わるフラグ変数の参照でも同様のことが起こります．これを防ぐには，

```
#define INFREG   (*(volatile unsigned char*)0xfff
f1234)
```

というふうにvolatile宣言をして，コンパイラにこの変数に対して最適化しないよう指令します．

注1：たいていそれは実験室の中ではなく，出荷後の顧客製品において起こる．
注2：よくある例は，割り込みフラグなどである．

たち・のぶゆき
NECマイクロシステム（株）